Actividad 6

Aspirar una habitación

En esta actividad vamos a programar un robot aspiradora para limpie todo lo posible el suelo de una habitación y debe saber cuándo ha terminado su tarea. Realmente lo que programaremos, ya que el Pioneer no tiene aspiradora, será que el robot pase por "todos los lugares alcanzables" de una habitación, las comillas están para indicar un requisito teórico, en la práctica bastará con que lo haga por casi todos.

Para realizar esta tarea el robot va a disponer de un mapa junto con un algoritmo de localización y otro de planificación de trayectorias. Para equipar al robot con estas nuevas características vamos a usar nuevas herramientas de programación, además de la librería Aria vamos a usar la librería Arnl y el programa cliente gráfico MobileEyes.

6.1. Instalación del nuevo software

El nuevo software se halla disponible en Alf en la carpeta Software externo.

6.1.1. Descarga

El principal es la librería de localización usando medidas láser, Arnl . Arnl se compone de dos librerías: libBaseArnl.so y libArnl.so, hay que descargar la versión 1.9.4 (1.9.3 en Windows) de cada una. Si usáis Debian 64bits podéis instalar los paquetes de Ubuntu.

MobileEyes es un cliente para comunicarse con el robot a través de un servidor usando una interfaz gráfica de usuario.

6.1.2. Instalación

El procedimiento de instalación se detalla en los enlaces anteriores. MobileEyes es un programa de 32 bits, por lo que es necesario tener activada la multiarquitectura, más información MultiArch.

6.1.3. Prueba

Una vez instalados los paquetes deberemos arrancar MobileSim con el mapa office3.map, disponible en los Documentos de Alf en la carpeta mapas simulador. A continuación ir al directorio /usr/local/Arnl/examples y ejecutar desde consola ./arnlServer -map <path-a-office3.map>. Ejecutar MobileEyes, se accede a él desde el menú de aplicaciones o directamente /usr/local/MobileEyes/bin/MobileEyes y en la pantalla de inicio conectar al Robot Servers localhost, no son necesarios ni usuario ni contraseña. Clicar en Tour goals y el robot empezará a calcular paths a goals y seguirlos, visitando todos los goals en un orden determinado. MobileEyes también permite mover el robot con las flechas del teclado o con un widget, ver figura 6.1.

arnlServer actúa como servidor entre el simulador o robot real y clientes como MobilesEyes. Tiene una particularidad, guarda la posición del robot la última vez que se utilizó el programa. Al arrancarlo con el simulador colocará el robot en esa posición. Sin embargo si el simulador se ha cerrado y vuelto a arrancar esto suele provocar la deslocalización del robot, basta con ver en MobileEyes que las mediciones y los obstáculos no coinciden para nada. Para solucionarlo se puede eliminar el fichero de texto <directorio-de-ejecución-del-arnlServer>/robotPose antes de arrancar arnlServer.

Consultar la documentación de MobileEyes y experimentar un rato con él como actividad previa. Por ejemplo observar casos en los que el algoritmo de localización funciona correctamente, figura 6.1 y otros en los que falla bastante, figura 6.2, y cómo se corrige la mala localización. Habilitar en MobileEyes View->Custom details para ver la probabilidad de localización.

6.2. Plantilla para QtCreator

El equipo docente ha creado una plantilla para QtCreator a partir de la cual desarrollar la actividad de este tema. La plantilla se denomina plantila_arnl, disponible en los Documentos de Alf en la carpeta Software, plantilla_arnl.tar.gz. Se compone de main.cpp, que es una versión simplificada de arnlServer, hemos eliminado lo que no se usa, y de la clase Control, que es en la que hay que trabajar para la actividad.

Para establecer el parámetro del fichero de mapa en el ejecutable desde QtCreator hay que proceder como se indica en la figura 6.3.

La clase Control recibe en su constructor punteros al robot y las las clases ArPathPlanningTask y ArLocalizationTask. Con el puntero al robot podemos controlarlo como de costumbre, de ArLocalizationTask solo recibimos un callback, Control::locFailed(), para que nos avise e muestre un mensaje si ocurre la tarea de localización falla, contra esto nada podemos hacer salvo empezar de nuevo, no es frecuente que ocurra en los mundos que usaremos. Los callbacks más interesantes son los que manda ArPathPlanningTask, para notificar si se ha llegado al goal (Control::goalDone(), si no (Control::goalFailed()), cuando ArPathPlanningTask cambia de estado (Control::pathPlanStateChanged()) y para otras situaciones infrecuentes (Control::goalInterrupt()).

La función Control::execute() es llamada cada ciclo del robot (100 ms.). En execute() se ordena planificar y seguir la ruta a la nueva pose y en ella se ha introducido un código para

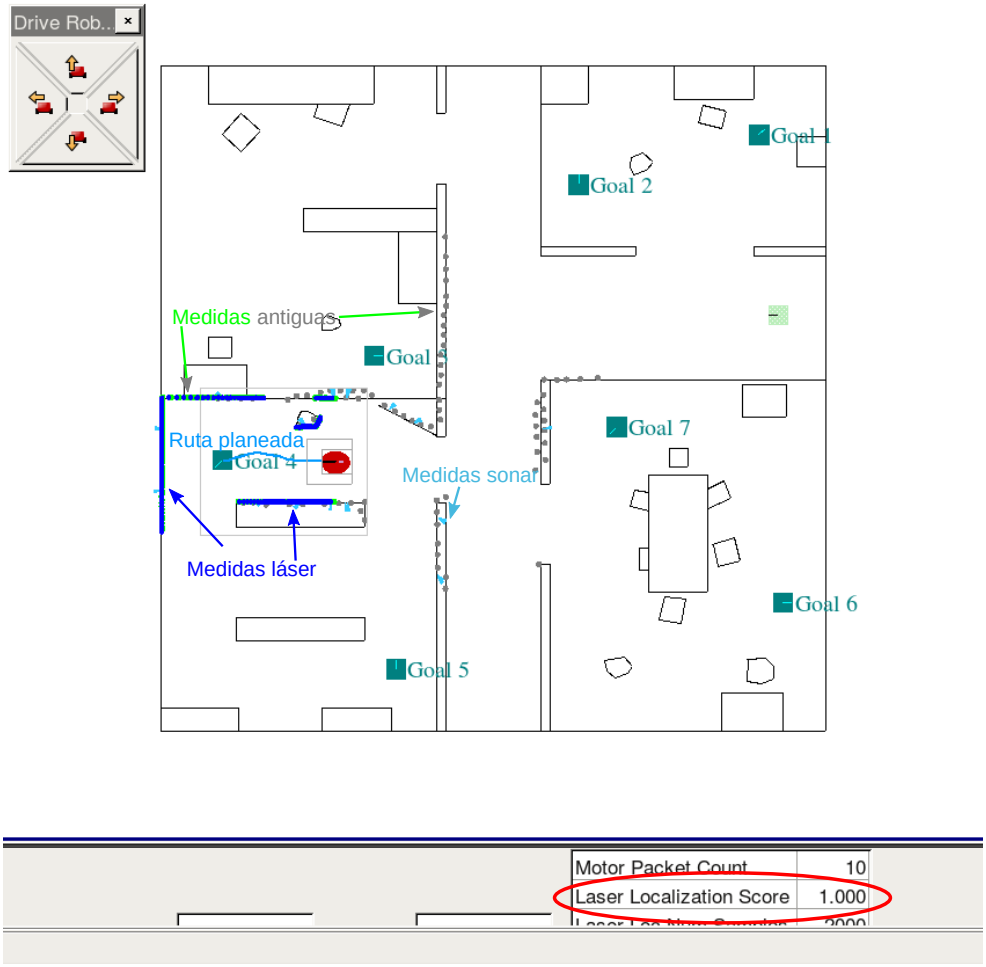

Figura 6.1: Vista de MobileEyes, caso de una muy buena probabilidad de localización. Las medidas láser coinciden prácticamente con los obstáculos del mapa.

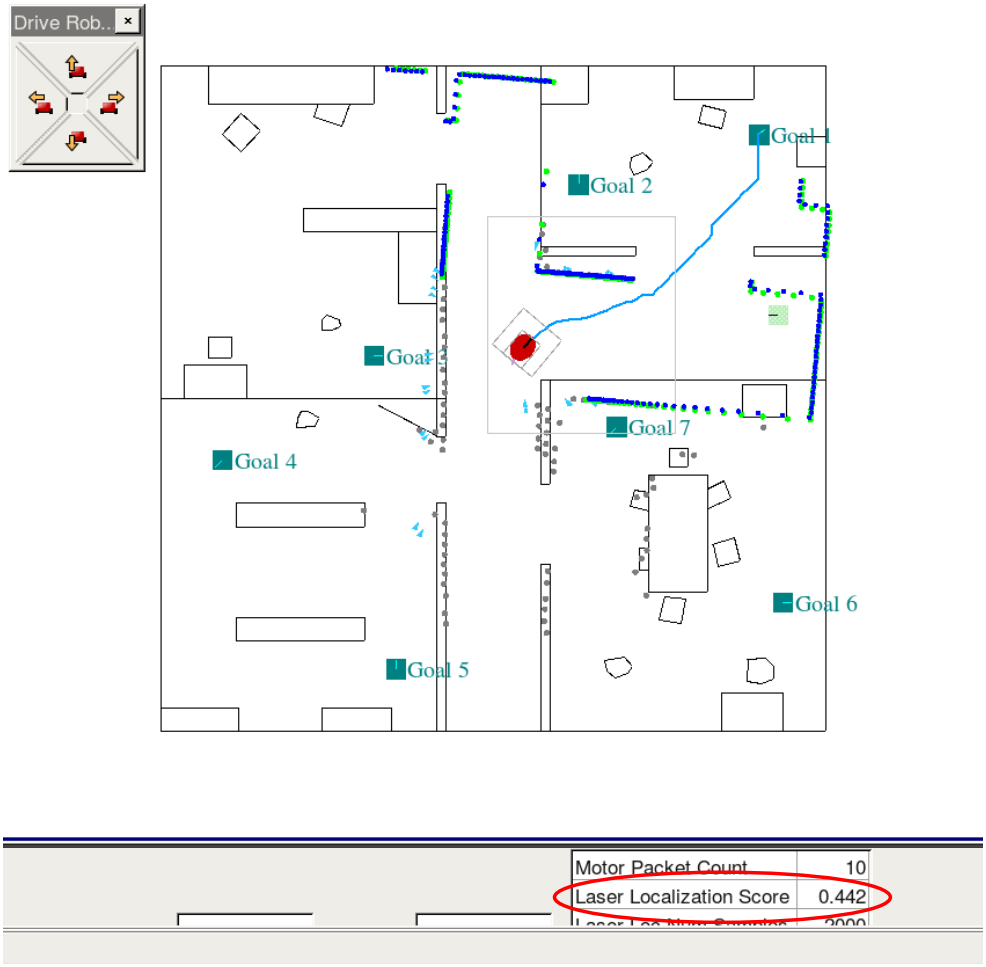

Figura 6.2: Vista de MobileEyes, caso de una mala probabilidad de localización. Las medidas láser están desplazadas de los obstáculos del mapa, debido a que el robot no está donde cree.

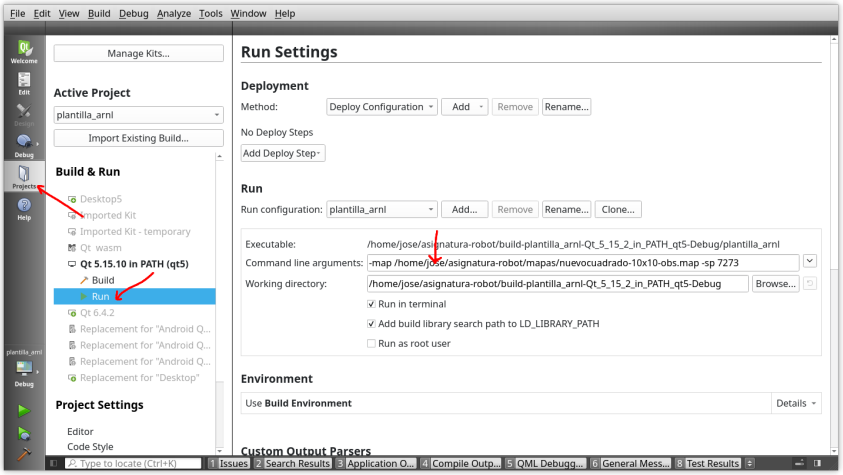

Figura 6.3: Establecer el fichero de mapa en el ejecutable

efectuar movimientos cortos directos, mediante Command Functions, en situaciones especiales, como por ejemplo varios fallos consecutivos de alcance de goals sin que el robot se mueva apreciablemente.

En el ejemplo los callbacks generan un mensaje y una pose nueva aleatoria dentro del mapa, otros solo tienen el mensaje. La primera pose no es aleatoria sino que se sitúa un metro a la derecha del Home, para que se vea el efecto de la cercanía de la pared en el cálculo de la trayectoria, ésta no es recta. Al dispararse goalFailed() se dispara, como ejemplo, un movimiento hacia atrás de 500 mm. En los comentarios del código se puede ver cómo realizar este tipo de movimientos cortos sin interrumpir el ciclo del robot.

6.3. Actividad

La actividad va a consistir en comenzando aproximadamente en la esquina superior izquierda del mapa "pasar por encima de todas las posiciones del mapa", consideramos que la hipotética aspiradora ocupa el ancho del robot. El mapa será nuevocuadrado-10x10-obs.map , disponible en los Documentos de Alf en la carpeta mapas simulador.

Hacemos la simplificación de comenzar en una posición en la esquina superior izquierda y saberlo, para simplificar un poco el problema, y es válido conocer las dimensiones del mapa como se hace en Control::generateRandomPose(), pero ninguna otra información sobre el mapa de la clase ArMap ni similares.

6.3.1. Grid

Una manera de llevar a cabo la tarea consiste en definir una rejilla (grid) que cubra el mapa e ir intentando desplazarse de celda en celda o cada dos o tres celdas y ver si podemos ocupar la celda correspondiente, colocaríamos un goal en el centro de la casilla correspondiente y se

intentaría alcanzar. Las celdas por las que se pasa se marcan como libres y aquellas a las que no se puede acceder se marcarían como inaccesibles. Bastaría con tener un array bidimensional de booleanos para ir consignado la accesibilidad de las celdas.

A las celdas les podemos dar aproximadamente el tamaño del robot, en la figura 6.4 tienen 50x50 cm. La trayectoria que se sugiere en la figura intenta pasar por cada celda, minimizando el número de veces que se pasa más de una vez por una casilla determinada, o sea, intentando minimizar la longitud de la trayectoria.

No intentaremos pasar por las casillas pegadas a las paredes exteriores de la habitación, como se ve en la figura, debido a que el procedimiento de seguimiento de trayectorias considera esas casillas demasiado próximas a obstáculos, como se puede comprobar mandando al robot a una de esas casillas. Las casillas a las que nos referimos son las de la primera y última fila y las de la primera y última columna, para el resto de casillas habrá que comprobar si son alcanzables o no.

Hay goals que ArPathPlanningTask considera inválidas sin necesidad de mover el robot, p. ej. las que están ocupadas por obstáculos. A otras planea la trayectoria pero cuando está llegando no puede alcanzarlas, en muchas ocasiones debido a errores de localización. Para reconocer primer tipo de goal se puede comprobar que el robot apenas se ha movido cuando se invoca a goalFailed(). Para el segundo se anotaría la posición de parada del robot y se usaría como goal asociado a la celda, en lugar de su centro.

6.3.2. Grabación y automatización de la ruta

Una vez visitadas todas las casillas, se debe guardar un archivo con la matriz de celdas alcanzables escritas en el orden en el que se siguieron en la exploración, en las clase Control hay un ejemplo del uso de las clases QFile y QtextStream para escribir en ficheros.

En sucesivas ejecuciones del procedimiento se procederá a leer el fichero de celdas y seguir los goals en el orden que aparecen, de esta manera se automatiza la tarea. Poned la opción en el programa de que si el fichero existe entonces se lea y se proceda a seguir la trayectoria definida en él.

6.3.3. Recomendaciones

Realizad experimentos con el código del equipo docente para explorar la casuística que se puede presentar. Así veréis situaciones en las que el robot está un tanto deslocalizado y no se atreve a moverse porque está muy cerca de obstáculos, estas son las situaciones en las que habría que emplear alguna táctica para escapar de la situación. Para estas situaciones tenemos los Command Functions en Control::execute(). La implementación que hemos realizado solo es un ejemplo ingenuo, en un caso real convendría mirar los sensores para decidir hacia donde realizar el movimiento de escape.

Tened en cuenta los mensajes que se reciben en Control::pathPlanStateChanged(), tiene información relevante acerca de la situación del goal actual, investigad estos mensajes con el código del equipo docente en distintos mundos.

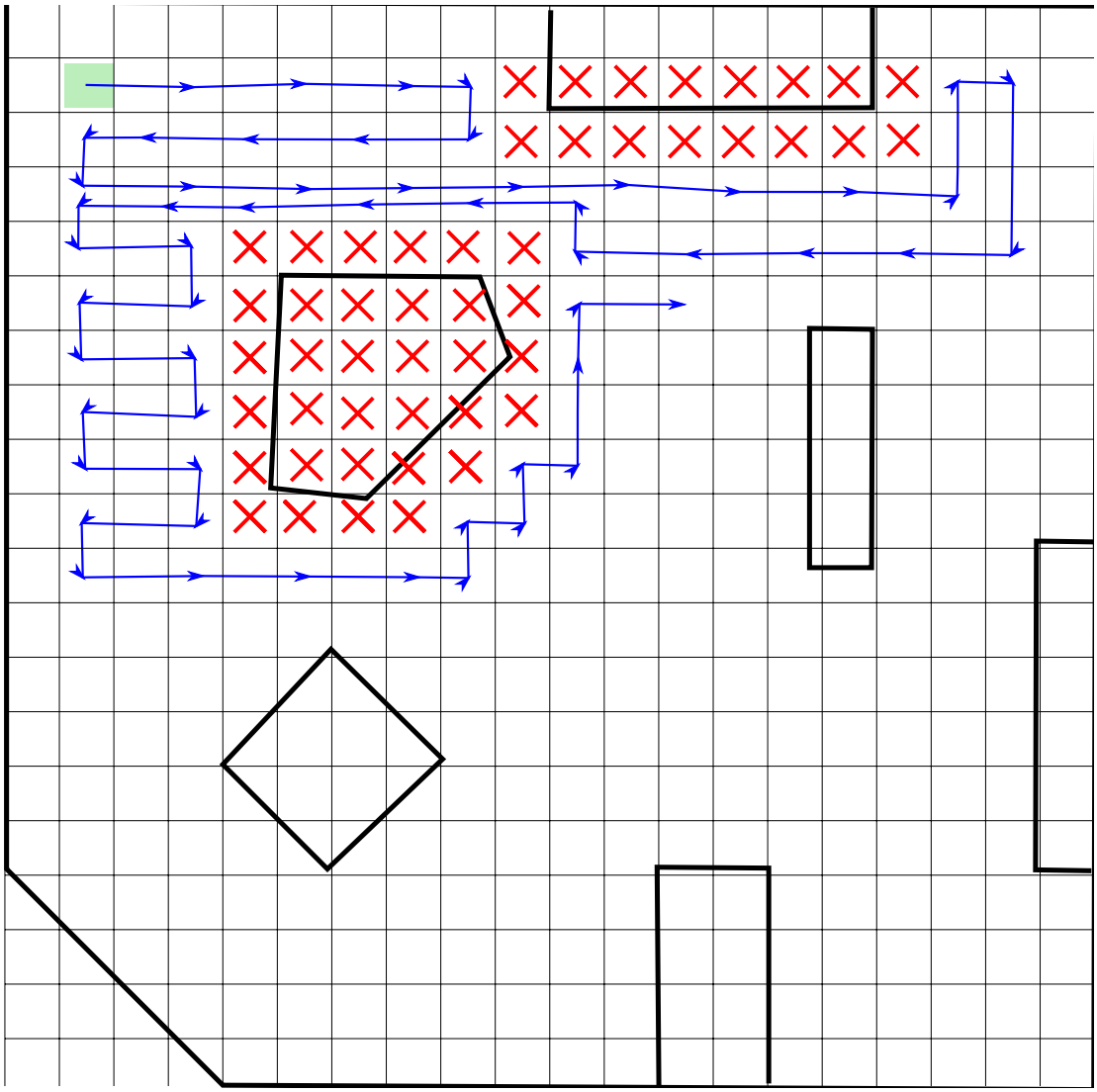

Figura 6.4: Sugerencia de procedimiento para explora las celdas del mapa. No se han marcado como ocupadas las casillas de las primera y últimas filas y columnas, aunque el robot no puede alcanzarlas, por simplicidad. Ya se comentó que se consideran ocupadas a priori.

Probar primero a realizar la tarea en el mapa cuadrado6x6.map, que no tiene obstáculos una vez que consigáis mover el robot siguiendo el grid pasar a nuevocuadrado-10x10-obs.map, e intentad diseñar un procedimiento que vaya pasando por todas las casillas, salvo las comentadas de los bordes del grid, intentando minimizar el número de veces que se pasa por una casilla determinada.

No os dejéis engañar por la vista de pájaro que se tiene cuando un humano ve el mapa, pensad en que sois seres 2D pegados al plano del mapa y que queréis recorrerlo, no deis nada por supuesto, sobre ocupación del celdas, salvo lo que se pueda comprobar con los procedimientos que diseñéis.

Algunas lecturas sobre el uso de grids en localización y generación de mapas, el tema que nos ocupará la última actividad del curso tutorial mapping and localization y para profundizar survey Sebastian Thrun.